REMARKS

Claims 1, 3, 7, 10, 11, 14-20, and 27-31 remain pending in the present application. Claims 3 and 7 have been amended to change "consisting" to "comprising." No other claim has been amended. A typographical error was corrected on page 3 of the specification. No new matter has been added to the application by the amendments.

In the Official Action, claims 1, 3, 7, 10, 11, 14, 15, 18-20, and 27-29 stand rejected under 35 U.S.C. § 103(a) as allegedly being obvious over You (U.S. Patent No. 6,158,045) in view of Niemi et al. (U.S. Patent No. 6,470,388). Claims 16, 17, 30, and 31 stand rejected under 35 U.S.C. § 103(a) as allegedly being obvious over You and Niemi et al. further in view of Hawley et al. (U.S. Patent No. 5,533,192). These rejections are respectfully traversed.

Independent claim 1 recites a debugger for debugging any of a plurality of debuggees. Each debuggee has a debugging type attribute selected from a plurality of debugging type attributes and representative of a type of debugging to be performed with respect to the debuggee, and each debuggee also has a processor attribute selected from a plurality of processor attributes and representative of a type of processor associated with the debuggee. The debugger is instantiated on a computer and has a single debugger engine for performing debugging functions with respect to any of the plurality of debuggees.

The engine includes a plurality of debugging type blocks, where each debugging type block supports at least one of the plurality of debugging type attributes, and a plurality of processor blocks, where each processor block supports at least one of the plurality of processor attributes. A particular debugging type block and a particular processor block are

selected for debugging a particular debuggee based on the debugging type attribute and processor attribute of the particular debuggee.

Significantly, the plurality of debugging type blocks are organized into a debugging type abstraction available to provide debugging type services that vary in implementation for each debugging type. The debugging type abstraction comprises programming code, and at least a portion of the programming code for the debugging type abstraction is common as between at least some debugging type blocks and is shared by such debugging type blocks. In particular, the programming code for the debugging type abstraction is organized into a tree form with generic code at a base node and more specific levels of code branching out at nodes therefrom. The debugging type abstraction nodes include leaf nodes from which no other nodes branch out. Significantly, each debugging type block is defined to include a plurality of nodes extending from the base node to a particular leaf node.

Likewise, the plurality of processor blocks are organized into a processor abstraction available to provide processor services that vary in implementation for each processor. As with the debugging type abstraction, the processor abstraction comprises programming code, and at least a portion of the programming code for the processor abstraction is common as between at least some processor blocks and is shared by such processor blocks. Also as with the debugging type abstraction, the programming code for the processor abstraction is organized into a tree form with generic code at a base node and more specific levels of code branching out at nodes therefrom. As with the debugging type abstraction, the processor abstraction nodes include leaf nodes from which no other nodes branch out. Significantly, each processor block is defined to include a plurality of nodes extending from the base node to a particular leaf node.

Independent claim 20 recites subject matter similar to that of claim 1, albeit in the form of a computer with the debugger instantiated thereon.

As was previously pointed out, the You reference discloses a debugger built on an object-oriented programming framework that is portable to multiple operating systems and hardware platforms. You teaches with respect to Figure 9 an inheritance graph for sample processor classes in an address class hierarchy. However, as acknowledged by the Examiner, You does not teach a debugging type abstraction hierarchy as claimed. In particular, You does not disclose or suggest a debugger with:

> a plurality of debugging type blocks, each debugging type block for supporting at least one of the plurality of debugging type attributes;
> ...
> wherein a particular debugging type block and a particular processor block are selected for debugging a particular debuggee based on the debugging type attribute and processor attribute of the particular debuggee,
> wherein the plurality of debugging type blocks are organized into a debugging type abstraction available to provide debugging type services that vary in implementation for each debugging type,
> wherein the debugging type abstraction comprises programming code, and
> wherein at least a portion of the programming code for the debugging type abstraction is common as between at least some debugging type blocks and is shared by such debugging type blocks,
> wherein the programming code for the debugging type abstraction is organized into a tree form with generic code at a base node and more specific levels of code branching out at nodes therefrom, the debugging type abstraction nodes including leaf nodes from which no other nodes branch out, each debugging type block being defined to include a plurality of nodes extending from the base node to a particular leaf node,

as required by independent claims 1 and 20. For such teachings, the Examiner turns to Niemi et al. However, Niemi et al. do not suggest the claimed features either.

Niemi et al. disclose a system that utilizes "debug" objects as illustrated in Figure 4. Debug sub-class 416 is used to define one or more debug objects 418 for use in debugging an application or process. Each debug object has a level that provides a different granularity of

debugging information or control. However, Niemi et al. do NOT teach that the different levels of debug object comprise programming code wherein:

> at least a portion of the programming code for the debugging type abstraction is common as between at least some debugging type blocks and is shared by such debugging type blocks,
> wherein the programming code for the debugging type abstraction is organized into a tree form with generic code at a base node and more specific levels of code branching out at nodes therefrom, the debugging type abstraction nodes including leaf nodes from which no other nodes branch out, each debugging type block being defined to include a plurality of nodes extending from the base node to a particular leaf node,

Applicants respectfully submit that Niemi et al.'s teachings of each level of debug object providing a "different granularity of debugging information or control" does NOT teach or suggest that the different debug objects share programming code organized into a tree and leaf nodes as claimed or that such code is shared by the different debug objects accessible by debugging type attribute as claimed. Applicant can find no such teachings or suggestions by Niemi et al., and the Examiner has not pointed to any.

Accordingly, even if the teachings of Niemi et al. could have been combined with the teachings of You as the Examiner suggests, the claimed invention would not result as neither You nor Niemi et al. discloses or suggests that a debugging type abstraction, accessible by specifying a debugging type attribute, may be employed and structured in the manner recited in claims 1 and 20, and that a corresponding processor abstraction likewise be employed and structured in the manner recited in claims 1 and 20. Accordingly, the combination of the You and Niemi references does not result in the subject matter recited in claims 1 and 20. Applicants respectfully request reconsideration and withdrawal of the rejection of claims 1, 3, 7, 10, 11, 14, 15, 18-20, and 27-29 over the teachings of You and Niemi et al.

The teachings of You are also deficient with respect to the claimed tree form. The Examiner once again points to Fig. 9 of the You reference as showing the claimed tree form. However, Fig. 9 shows an addressing abstraction utilized to facilitate the use of target memory addresses in a portable fashion (Abstract), and only a particular one of the nodes within such addressing abstraction is selected to locate a particular address, depending on operating system and/or platform. In contrast, the present invention as recited in claims 1 and 20 requires that the programming code for both the debugging type abstraction and the processor abstraction is organized into a tree form with generic code at a base node and more specific levels of code branching out at nodes therefrom such that each respective block is defined to include a plurality of nodes extending from the base node to a particular leaf node. Moreover, the tree of Fig. 9 of You has a plurality of nodes, each for a particular type of address, where selecting a particular node is necessary for selecting the corresponding type of address. In contrast, the tree recited in claims 1 and 20 has a plurality of nodes where a particular debugging type block or processor block is defined by selecting a plurality of such nodes extending from a base node to a particular leaf node, as may best be appreciated with reference to Figs. 4 and 5 of the present application. Contrary to the Examiner's assertions, such distinctions are not taught by You.

With respect to claims 16, 17, 30, and 31, the Examiner further acknowledges that You does not disclose that the executable code includes an attribute for use in selection of a particular debugging type block in the engine. For such a teaching, the Examiner cites Hawley et al. for Hawley et al.'s teaching of the selection of one of multiple debuggers. However, as with You and Niemi et al., the teachings of Hawley et al. are deficient in that Hawley et al. also fail to teach debugging type abstraction, accessible by specifying a

debugging type attribute, may be employed and structured in the manner recited in claims 1 and 20, and that a corresponding processor abstraction likewise be employed and structured in the manner recited in claims 1 and 20. Absent such teachings, even if the cited references could have been combined as proposed by the Examiner, the claimed invention would not have resulted. Moreover, the Examiner has provided insufficient teachings, suggestions, or motivations for combining the teachings of the cited references as proposed to arrive at the claimed invention. In view of these deficiencies, the Examiner has not established *prima facie* obviousness and the obviousness rejection must be withdrawn.

The Examiner is reminded that, as set forth in M.P.E.P. §§2142-2143.03, in order to establish a prima facie case of obviousness, patent examiners are required to establish three criteria: (1) there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings; (2) there must be a reasonable expectation of success; and (3) the prior art reference, or combination of references, must teach or suggest all the claim limitations. The examiner bears the initial burden of factually supporting any prima facie conclusion of obviousness. To make a proper obviousness determination, the examiner must "step backward in time and into the shoes worn by the hypothetical 'person of ordinary skill in the art' when the invention was unknown and just before it was made." In view of the available factual information, the examiner must make a determination as to whether the claimed invention "as a whole" would have been obvious at that time to a person of ordinary skill in the art. Importantly, a rejection based on these criteria must be based on what is taught in the prior art, not the applicant's disclosure. The applicant's disclosure may not be used as a blueprint from which to construct an obviousness rejection.

The You, Niemi et al. and Hawley et al. patents, taken together, do not teach or suggest, together or in any proposed combination of teachings, all of the elements of claims 1 and 20 and thus do not establish *prima facie* obviousness. Withdrawal of the obviousness rejections and allowance of claims 1 and 20 and all claim dependent thereon is respectfully requested.

Conclusion

In view of the foregoing, Applicants respectfully submit that the present application including claims 1, 3, 7, 10, 11, 14-20, and 27-31 is in condition for allowance, and a Notice of Allowability is respectfully requested.

Date: December 26, 2006

/Michael P. Dunnam/
Michael P. Dunnam
Registration No. 32,611

Woodcock Washburn LLP
One Liberty Place - 46th Floor
Philadelphia PA 19103
Telephone: (215) 568-3100
Facsimile: (215) 568-3439